AFIT/GCS/ENG/97D-07

Concept Vectors: A Synthesis of Concept Mapping and Matrices for Knowledge Representation in Intelligent Tutoring Systems

THESIS
Mark L. Dyson
Captain, USAF

AFIT/GCS/ENG/97D-07

AFIT/GCS/ENG/97D-07

Concept Vectors: A Synthesis of Concept Mapping and Matrices for Knowledge Representation in Intelligent Tutoring Systems

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Mark L. Dyson, B.S.
Captain, USAF

December 16, 1997

AFIT/GCS/ENG/97D-07

Concept Vectors: A Synthesis of Concept Mapping and Matrices for Knowledge Representation in Intelligent Tutoring Systems

Mark L. Dyson, B.S.

Captain, USAF

Approved:

[illegible]	18 Nov 97
Dr. Eugene Santos, Jr.	Date
Thesis Advisor	
[illegible]	18 NOV 97
Lt. Col. Gregg Gunsch	Date
Committee Member	
[illegible]	18 Nov 97
Maj. Sheila Banks	Date
Committee Member	
[illegible]	18 Nov 97
Maj. Michael Talbert	Date
Committee Member	

Table of Contents

List of Figures

AFIT/GCS/ENG/97D-07

Abstract

A review of the literature relating to intelligent tutoring systems (ITS) reveals that the bulk of research to date is focused on the student, and on methods for representing the knowledge itself. From student models to learning schemas to presentation methods, comparatively little attention has been paid to the problem of educators attempting to build viable lesson plans for use in an ITS environment—yet when this problem is addressed in the literature, it is recognized as a potentially daunting one. This thesis addresses the problem of ITS lesson plan development by proposing a practical, computable approach for knowledge engineering that is based on proven classroom methods. The document then details a system for dynamically creating lesson plans from a knowledge base created under the described methodology, using already-established algorithms of proven tractability, and then discusses how this system can be integrated into existing and future ITS design.

Concept Vectors: A Synthesis of Concept Mapping and Matrices for Knowledge Representation in Intelligent Tutoring Systems

I. Introduction

Artificial intelligence has been applied to education for years; in fact, trend analysts from as far back as 1983 declared such application to be "inevitable(10)." Nonetheless, despite ever-growing focus on the computer as a dominant medium in the field of educational technology(3), expert opinion concerning the utility of artificially intelligent teaching tools ranges from statements that instructional programs "don't know what they're doing(10)" to the general conclusion that such programs are of poor quality(3).

Before an intelligent tutoring system (ITS) can be employed, it must have a knowledge base from which to teach. That knowledge must be represented in a tractable form to be useful–both from a computing standpoint and from the point of view of presenting that knowledge to a student.

In the literature one finds numerous examples of knowledge representation schemes, from the idea idea of *concept mapping*(9) to the intensive, heirachical databases used in the Air Force's Instructional System Development (ISD) project(5). Unfortunately, even in the case of automated tools such as provided in the ISD project, educators must face steep learning curves, a lack of a standardized interface, and a significant amount of manual development when constructing lesson plans(5).

1.1 Problem Statement

To date, the bulk of ITS research tends to focus on student modeling and knowledge presentation; in short, on the student's perspective within the learning environment. As pointed out in the ISD project, the educator, while attempting to develop lesson plans for an ITS environment, is often left with a significant amount of development work(5). What is needed, before attempting to design an ITS, is a methodology for defining and developing lesson plans in a form directly suitable to ITS implementation. This methodology needs to be quantifiable both in terms of content and applicability, and able to accept feedback metrics on a given student's progress in order to modify the lesson plan as needed. In addition, any tool providing this capability should be useful without requiring excessive training.

1.2 Assumptions

The design for the lesson plan generator assumes a nontrivial amount of knowledge engineering prior to implementation. Not only must the knowledge domain itself be quantified according to the procedures outlined herein, the material from which the generator is to draw must be defined within the context of that mapping effort; again, according to the methodology detailed in this thesis. It is further assumed that, as a heirarchical knowledge domain is broken down into its smallest practical components, those components (or concepts) are taught in a linear fashion, beginning with the student knowing nothing about the concept in question and leading to the goal of complete understanding. This perspective lends itself well to normalization of a student's progress in mastering a concept, and greatly simplifies the task of representation. Further, this assumption of a linear progression within a given concept reflects not only the views of educational theory over more than fifty years (such as Thorndike on "connectionism(12)" and Guthrie on the contiguity of learning(4)), but the arguments of more modern educational theorists who assert that not only does learning a concept rely on its context, but also that despite the ability to manipulate information granted by computer technology, human learning still ultimately comes down to step-by-step repetition within that context(6). Creating the concept map of a given kowledge domain and mapping lesson modules to that map lies outside of the scope of this research effort; therefore the algorithm presented assumes an already-established database of lesson modules from which to draw a sample lesson plan, created in accordance with the methodology described herein.

1.3 Scope and Approach

This thesis is intended to introduce a respected and long-standing method of representing complex knowledge domains in the classroom, and to illustrate a system for mapping that representation into computer-readable form. It further describes a system, based on that form, able to accept relatively simple inputs from an educator via a commonly-used interface and return a dynamic lesson plan. It will also discuss the feasibility of a direct interface between lesson plan development and an ITS, permitting seamless integration of educator input and student performance into follow-in lesson plan creation.

The lesson plan generator introduced here is intended to provide the educator with the means to generate dynamic lesson plans from an existing database of teaching materials using a simple world wide web (WWW) interface to JAVA-enabled WWW browsing software. Not only will the educator be able to request a lesson plan covering just the material of interest, he or she will be able to modify the request to account for the results of previous test scores so that satisfactorily-completed material can be excluded.

The thesis will begin by introducing the idea of concept mapping, provide an illustrative example of a concept map, and then show how this map is used by a knowledge engineer to define teaching materials within its domain. It will introduce concept vectors, motivate their usefulness in representing concepts in computer-readable form, and provide theoretical grounding for an effective algorithm for creating a lesson plan based on vector sums. It will also discuss implementation details, including samples from a prototype system, and then conclude with a discussion of how the prototype could be enhanced via future research.

1.4 Organization of Thesis

Chapter II consists of a review of the more influential items from the literature as applied to the research documented here. Chapter III details the methodology of constructing a lesson plan builder under the proposed representation scheme, including algorithm and a sample system. Chapter IV discusses the effects of the assumption of concepts being linear, proposes an approach based on that assumption being removed, and then touches on areas for future research, including automated interfaces between the lesson plan builder and an actual ITS.

II. Literature Review

The literature reviewed is mentioned in order of its applicability to the overall thesis organization. The first sources deal with education in general, with an eye to how they contributed to this research. The following section discusses sources specific to the development of the system itself.

2.1 Educational Theory

The sources described in this section relate to the education and knowledge engineering aspects of the research presented.

2.1.1 General Theory. Two papers written in the 1930s proposed theories on how people learn. In 1930, E. R. Guthrie published the "contiguity theory(5)" that asserted that once a learner has made a link between an action and learning from that action, repetition of that action is beneficial to the continued learning process. This assertion, as well as that made by E. Thorndike in 1932(20), states that not only does learning depend on new situations containing elements of earlier ones but also learning responses tend to chain together towards a goal. These assertions contributed to my representation of learning within a single concept as a linear progression, having a beginning and end point, and covering the entire range in between. Further, Guthrie(5) contends that instruction must present very specific tasks, which is in keeping with my emphasis on nodes within a concept map being as low-level and specific as possible.

Current theorists in the literature tend to bear out these earlier works, even in the current culture of computers in the learning process. Trend watchers such as D. Ely(4) and T. O'Shea(14) emphasize how the process of learning transcends computing, rather than is overshadowed by it, while educational researchers such as J. Brown(7) point out that underneath sophisticated computing tools designed to aid learning, the learning process itself has remained relatively unchanged.

2.1.2 Concept Mapping. Novak and Gowin's book *Learning how to Learn*(13) was perhaps the most influentual single piece of literature in terms of this research. The idea of concept maps is a powerful one, well grounded in current educational practice(2) as well as providing a framework for representing even complex knowledge domains in a disciplined and easily-computable fashion. The methods detailed in this book were key to describing the knowledge engineering effort required to quantify a knowledge domain, and provided the context for defining lesson modules. An example supplied in the book provided the domain partially implemented as a proof of concept in this thesis.

2.2 Algorithm Design and Theory

The sources described in this section relate to the computer science and implementaion aspects of the research presented.

2.2.1 General Algorithmics. While there exist numerous texts on general algorithm theory, two were of particular benefit in constructing the system described in this thesis. D. Kozen's monograph based on his years of lecturing at Cornell(10) provided an excellent foundation for understanding matroid theory, and how a given problem can be mapped to one of several classic algorithmic categories, providing a solid theoretical foundation for the algorithm selected in this research. In particular, Kozen's rigorous definition of matroids is used in this thesis to demonstrate that concept vectors are members of this class of data structures; the importance of this membership to manipulating concept vectors is discussed more fully in the next chapter. The idea of using scalar values to contain comples underlying information through binary coding is a venerable one; similar techniques (such as Binary Coded Decimal) were introduced well over a decade ago as solutions for a multitude of problems in early computer design(11).

2.2.2 Specific Algorithm Treatment. Neapolitan & Naimipour's text(12) was the source for the recursive *sum_of_subsets* algorithm used in a slightly modified form in this thesis. Although *sum_of_subsets* was discovered to be a poor choice for final implementation, the text's thorough discussion of the underlying principles facilitated identifying the problems described in the next chapter. The algorithm text by T. Cormen, et. al.(18) provided amplifying information as to the power of matroids and greedy algorithms, as well as the pseudocode for the implementation of the Set Covering Problem detailed in the final proof of concept.

2.3 Other Literature

Although a search of the literature turned up little geared specifically to the problem of the educator, one handbook published by the Air Force's Instructional Systems Division cited the challenge of creating lesson plans using automated tools as "extensive(6)." The handbook described problems encountered, such as the lack of standard methodology and non-intuitive interfaces requiring steep learning curves. These are the very problems this research is aimed at addressing.

Two texts provided minor foundational grounding; the first, an undergraduate text on linear algebra by J. Auer(3), was included to show basis for asserting that vectors can be added to form a resultant. The second text, an introductory-level treatment of artificial intelligence by P. Winston(22), set the scene for asserting the importance of knowledge representation schemes in knowledge-based systems—an underpinning of the focus of this research.

As stated earlier, literature searches turned up a marked dearth of material relating to the educator's problem in composing ITS lesson plans; the bulk of the material available centers on the problems of the teaching process itself. This trend is reflected in prior AFIT research, as well, as attention is focused on key issues such as the difficulty of authoring ITS content(8) and in maintinging accurate models of student behavior during the instructional process(19). Even Matthew Kabrisky's landmark paper, cited as the instigator for AFIT's foray into ITS research(19), focused on the quality of the classroom interaction itself(9). Other research works seem to have continued to follow this trend of focusing on the student. For example, Shute and Psotka's visionary paper detailing their view of the sum of all ITS research(16), although it mentions the concept of "curriculum" as an important node in their ITS model, glosses over the process of how that curriculum would be obtained, relegating it to a problem to be handled by domain experts(16). They do concede, however, that the domain expert's problem is a difficult one(16), which underscores the cautions offered later in this thesis concerning the importance and difficulty of the knowledge engineering effort expected to implement this research in a real-world system. Shute's bias towards student models and learning behavior becomes evident in previous research for the Air Force(17, 15), and sets the scene for the general climate in ITS research to date.

Other works had tangential influence in this research, in some cases through the decision not to pursue their course. Asahi et al.(21) introduced a novel class of treemaps which allows for dynamic sensitivity analysis of search results and presenting easy-to-read graphical feedback to the user. This technique, while intriguing, was not included in this research; the heirarchical domain in which the lesson plan generator searches for lesson modules is expected to remain static during the course of a search, so the paper's methods for dynamic weighting would not apply. There is room for further research into how Asahi's methods apply to the classroom environment itself, but that is an issue for presentation methods during ITS sessions, and outside the scope of this research. A yet-unpublished work by Ahlberg and Shneiderman(1) was of some interest during the devlopment of a prototype WWW interface for the lesson plan generator. Their work explored the problem of making discrete selections from a potentially wide range of choices with an eye to enabling the user to make rapid and accurate selections. Although of general interest in the human-computer interface arena, their work was deemed over-complex for implementation here; one of the goals of this research was to make the user interface as simple and intuitive as possible, which would preclude the user from being presented with enough complex information to make something as powerful as an alphaslider necessary.

III. Implementing Concept Vectors

3.1 Knowledge Engineering

To begin any knowledge-based system, one first needs a workable system for representing that knowledge(13). One very good represention for even potentially abstract knowledge domains was developed in the mid-eighties by Joseph Novak and Bob Gowin(9). Not only does this representation, called concept mapping, enable a domain expert to map knowledge in a heirarchical tree structure amenable to computer processing, it has the additional virtue of being a respected and powerful tool in the classroom environment itself(1).

3.1.1 Concept Mapping. Concept maps represent an approximation of the relevant concepts and propositions of a given knowledge domain(9), and their creation requires both domain expertise and experience with the concept mapping process. Figure A.1 (see appendix A) provides an illustrative example of a concept map created by a domain expert; in this case, an expert on the topic of Meat Science(9). The first step is to identify the major topic to be mapped; in the case of the example, the domain expert has decided that the overarching consideration for this domain is meat quality. This concept of quality leads to the concepts of metrics by which to determine this quality, hence the concepts of judging, and then criteria. From this beginning the relevant criteria follow, and below them are identified the various conditions under which those criteria are affected, such as an animal's age, environment, and feeding habits. In the final form, the domain has been subdivided into a map of 29 nodes, each node representing an atomic concept in the knowledge domain as determined by the domain expert.

This methodical breaking down of a potentially complex domain into progressively smaller conceptual components is key to the idea of representing knowledge as vectors. A vector is made of components, each representing a magnitude along a single dimension in n-dimensional space(2). In order for the vector to be meaningful, each component must exist entirely within its dimension, having no contribution to the magnitudes of other components in that vector. Carrying this analogue to the concept mapping domain, if a vector represents the entire knowledge domain then each concept within the overall concept map can be thought of as a component of that vector. Further, the knowledge engineer must take pains to ensure the concepts are broken down as much as possible to prevent overlap between then—just as vector components provide contribution only in their dimension.

Once a given concept is identified to the desired level of simplicity, the domain expert must draw on available experience to determine what constitutes mastery of that concept. In the meat science case, for example, there exists the concept of "grass" as relates to feeding of meat animals(9). The domain expert might conclude that there are a finite number of types of grass an herbivore can encounter, and that mastery of "grass" means that a student can correctly identify each type and relate its effect on an animal's health in relation to the other types. Given this conclusion, a metric has been established wherein knowledge of "grass" can be measured on a scale from total ignorance to complete mastery. Once such a metric has been established for each concept in the domain, it is normalized so that complete ignorance is represented by zero, and total mastery by one. Applying such a numerical measure to each of the concepts makes it possible to represent the concepts in a machine-readable form, and to do computations based on levels of mastery of the knowledge domain. Further, the ability to represent any given level of knowledge within the domain by a string of numbers–a vector–which follows known properties becomes a powerful tool for knowledge representation.

It might be noted that the links in a concept map can contain information, and that a concept can be perceived as having a heirarchical position within the map, with parent and offspring dependancies. While the links are a potential source of amplifying information about the concepts they connect, this avenue is not explored in the research documented here; rather, the methodology presented assumes that all necessary information is contained within the concept nodes themselves. Further research is encouraged to explore the utility of using link information in some fashion. Information about the heirarchical relationships between concepts is likewise not utilized. Unlike nodes in a conventional tree structure which are traversed according to some ordering scheme, concepts as presented by an ITS are expected to be taught as discrete components. This assumption is discussed further in the section covering algorithm selection.

3.1.2 Mapping Knowledge to Concept Vectors. Once a concept map has been established for a given domain, the available teaching material must be reviewed and defined within that context. The key to this process is again to visualize the n nodes of the concept map as dimensions in a vector space. The teaching material available—be it a pamphlet, a section of a chapter from a textbook, or a previously-coded instructional module from an existing ITS—must be matched against which node(s) of the concept map to which it relates. For example, in the meat science case, there might exist in the curriculum a textbook on animal nutrition. Within that text might be found a chapter on herbivores, with one section discussing grass and another detailing various grain feeds. In this case the chapter would be treated as two distinct units of instructional material, each relating to a different concept.

The second step is to determine how much of the appropriate concept a given unit covers. Recall that for each concept in the domain, the span from complete ignorance to full mastery is represented as a normalized scale from 0.0 to 1.0. As a unit is evaluated against a given concept, the domain expert determines what degree the concept is covered and assigns appropriate start and end values. It's important to reiterate here that teaching material is assumed to follow a progression from ignorance to mastery. The act of defining a unit of teaching material in this context assumes contiguous coverage from lower to upper bound within the span–the smallest unit of teaching material in this context. If it's determined a unit does, in fact, have gaps in coverage as defined by the domain expert, that unit should be further divided into smaller units until no gaps exist in a single unit—further illustrating the importance of making the initial concept map as granular as possible.

As an illustrative example, consider Figure 3.1, a subset drawn from from Figure A.1. Of the seven concepts depicted, the four highlighted represent the concepts in which we're interested.

To posit a trivial case in which a single unit of teaching material–a textbook, perhaps–is available, and provides full coverage of all four concepts. While not a particularly interesting example, Figure 3.2 shows a single concept vector defined in this four-dimensional domain.

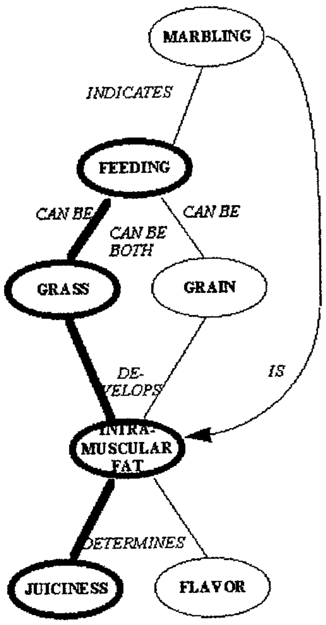


Figure 3.1 **Subset of Meat Science Course**(9)

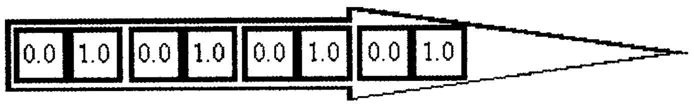


Figure 3.2 **A Single Four-Dimensional Concept Vector**

Note the presence of four numeric pairs, one for each concept represented. Within the pair each number represents the lowest and highest level of information provided by the unit of instruction as evaluated against the concept map and the normalized scale established for that particular concept.

A more interesting example is reflected in Figure 3.3, which shows three lesson modules represented as three concept vectors. In this case, the three vectors each only only address concept one; the first represents an entry-level module covering the concept from no prior knowledge to a point defined to be three-tenths of the entire range. The second module covers the range from two-tenths to eight-tenths, representing an intermediate level of information, while the third is the most advanced of the three, beginning seven-tenths of the way along the scale and covering the material up to total master of the concept. Note that, while no one of the three vectors shown covers the whole concept, the three taken together do provide complete cover.

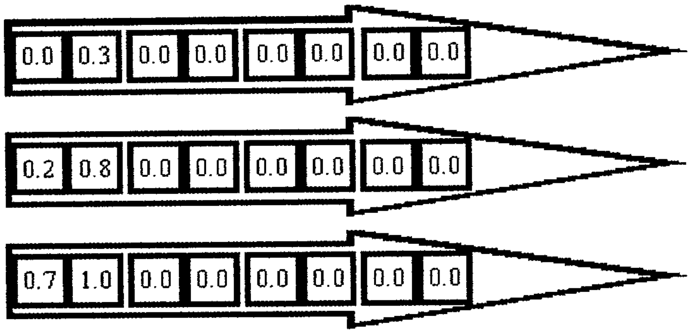


Figure 3.3 **Three Defined Concept Vectors**

There is some overlap in the depicted vectors; not all lesson materials available to an educator can be assumed to fit together without some overlap in how they cover a concept. Further, given effective granularity in the concept map definition, one can reasonably expect to see the typical vector providing a contribution in only one concept; in the case of the provided example the three units of material defined as vectors might represent three subsections of a textbook chapter on feeding techniques, with overlap reflecting review and cross-referencing by the author.

With the knowledge domain defined as an n-dimensional vector space, and with the available teaching materials defined within that space, the power of this representation becomes clear: the units of teaching material can be summed as vectors either to evaluate the completeness of available material within the context of the overall domain, or in order to reach a desired target vector. In the latter case, the units selected to sum to the target vector map directly to the teaching materials the educator needs to comprise a lesson plan designed to teach given concepts to a desired level.

At this point it is useful to re-visit the forms which lesson modules could take. Previous discussions have mentioned such items as pamphlets and sections from texbooks, but in practice such sources might be difficult to quantify as precisely as described in this thesis; they are mentioned mainly for illustrative purposes. It is expected that the design methodology presented will most useful when interfacing with an established ITS, presumably with a database of lesson modules already defined and coded. In such a case the knowledge domain is already defined by the material the ITS is designed to teach, and lesson modules will already be defined in terms which the ITS can use to differentiate between them for selection. Given this situation the KE effort becomes less daunting, and the goal of defining discreet concepts which map back to specific subsets of lesson modules intuitively becomes an easier one to attain.

3.1.3 Using Concept Vectors. The implementation vehicle selected for this research is the Java programming language. The intent was to insure portability of the implemented code across multiple platforms, and open up the possibility for creating an intuitive user interface to reduce the burden of learning to use the system. WWW browsers are in widespread use, provide a familiar graphical user interface (GUI), and Java-compatible versions are available for nearly every computer system available to today's educator. In a full implementation, an educator would be able to visit a lesson plan resource page with a Java- enabled WWW browser and select a desired knowledge domain. He or she would then be offered a representation of the concept map defined for that domain, and be allowed to select desired concepts from the map, as well as the desired lower- and upper bound for the coverage level for each concept to be taught. This selection would then define a "target vector" representing the level of knowledge the educator desires to impart in each of the appropriate concepts. Using the summation property of component vectors(2), the system would return a set of titles listing the appropriate lesson materials, in order, which will cover the topics in question—in other words, a subset of the vectors in the database which will sums to the target vector.

In this context the idea of a "best" (or optimal) solution is of interest. The educator might be interested in finding the smallest set of modules which cover the desired material, or might be operating under a set of constraints wherein modules from a certain source or possessing some other attribute are deemed more desirable than others. These issues are discussed in greater detail later in this chapter, as well some methods for achieving a "best" solution.

Responding to feedback derived from test scores is also possible using this system. Since the teaching materials are already defined in terms of the domain's concept map, it is straightforward to map testing results against the original lesson plan. By adjusting the lower bound of the coverage of a given concept upward (for example) to exclude material successfuly tested, the educator can create an updated lesson plan covering only that material the student failed to demonstrate mastery of, based on the test results.

3.2 Algorithm

The criterion for algorithm selection was to use currently defined algorithms throughout, rather than to "re-invent the wheel." The details of the algorithm selection, along the the related theoretical grounding, are discussed later in this chapter.

3.2.1 General Algorithm Considerations. The first step was to identify the general form of the problem. As with many schemes involving search, search space pruning and early arrival at optimal solutions is both desirable and often difficult(13). Within the context of this research, "optimality" is defined as the lesson plan generator consistently selecting a lesson module providing the largest overall amount of coverage (which still lies completely within the target range) whenever such a choice exists. Such choices will have the side benefit of tending to cover the target range using the fewest number of lesson modules possible, though in the implementation presented situations can be contrived wherein the number of modules selected is not minimal. The decision as to what is "optimal" is variable; a desired result might be to use as many modules as possible, so long as each provides at least some unique coverage within the range. The definition might be based on some metric defining quality of the lesson modules, or recency, or something else entirely. The important point to keep in mind is that "optimality" must be defined using some metric, and that a given solution can be called "optimal" only within the context of that metric. In this thesis the term "optimal" is assumed to conform to this restrictive context.

One approach to finding optimal solutions is the so-called greedy approach (7, 8, 11). A greedy algorithm selects the "best" choice (based on some metric) from the list of feasible candidates in a list until it arrives at a solution. The selection is typically made by sorting the candidates by the optimality metric, in nonincreasing order of desirability, so that the selection process is reduced to examining the list in order and selecting viable members until the solution is satisfied or found to be unreachable. By the definition of the desirability metric, each candidate so examined is the most valuable one yet unexamined. The benefit of this approach is that once a solution is found the algorithm terminates without searching for alternative solutions. Unfortunately, although greedy algorithms tend to converge to workable solutions, they can't always be guaranteed to find an *optimal* solution(11). The key to insuring optimal solutions with a greedy algorithm is in using the *matroid* property, which Kozen defines as follows(7):

matroid A pair $(S, \Im)$ where S is a finite set and $\Im$ is a family of subsets of S such that

(i) if $J \in \Im$ and $I \subseteq J$, then $I \in \Im$;

(ii) if $I, J \in \Im$ and $|I| < |J|$, then there exists an $x \in J - I$ such that $I \cup \{x\} \in \Im$;

In less rigorous terms, matroids are set structures having the property that subsets can be further broken down into smaller subsets which are still members of the original set, and it's possible to tranfer members from one subset to another without leaving the original set(11). The chief benefit from a search space being a matroid is that there exist numerous cases of greedy algorithms which have been proven to find optimal solutions for matroids(11).

To show that concept vectors are, in fact, matroids, one needs to demonstrate that both properties from the definition above hold(7):

Property (i) is straightforward to demonstrate: take the family of subsets of vectors from the database of concept vectors, S, and call it $\Im$. From that $\Im$ take a subset J, and then from J draw a subset I. It's clear to see that I was drawn directly from the family $\Im$ and therefore $I \in \Im$ holds. Property (ii) is similarly straightforward: since I, J are both drawn from $\Im$, then if I has a smaller cardinality (fewer number of vectors) than J, then there will exist some vector x from S which is in J but not in I. If you add that x to the set I, the resulting set will still be a subset of vectors drawn from S, and therefore $I \cup \{x\} \in \Im$ holds.

To conclude the general algorithm discussion, it can be shown that concept vectors are in the family of combinatorial structures known as matroids, and therefore an optimal solution can be guaranteed (within the context of how the greedy selection is made) using a greedy algorithm.

3.2.2 Specific Algorithm Considerations. Before examining the specific algorithms used, it is important to establish a assumption common throughout the following discussion: vectors are selected only according to their contribution to the dimension currently being evaluated. The primary effect of this assumption is that the definition of optimality defined for this research is only valid within the context of a single dimension; no global checking is performed to determine how selecting a module affects coverage in other dimensions. The rationale behind this simplifying assumption is that concepts, just as the lesson modules which pertain to them, are expected to be taught as a unit. That is to say that a given ITS, faced with teaching concepts B, D and E, is expected to present those concepts as distinct entities in some ordered fashion and not mix them. In similar fashion, the lesson plan generator will first select the set of modules to cover concept B, then a distinct set designed to cover concept D and, finally, concept E. The resulting list may contain modules which were coincidentally selected more than once, for contributions in more than one concept, but the ITS is expected to be presenting the material according to its concept map and so the distinction becomes unimportant. Furthermore, in the context of the expected close correlation between lesson modules as defined for the ITS and for the lesson plan generator, it is expected that lesson modules having contribution in more than one concept would be encountered rarely, if at all.

3.2.2.1 Failed Attempt - Sum Of Subsets Algorithm. Once it was determined that a greedy approach is suitable, the choice of the specific algorithm to use was based on insight into the problem itself. The system is being presented a desired target vector with components constrained within a finite range, has knowledge of a set of candidates each of which can provide some degree of contribution towards meeting the target vector, and asked to compose an optimal subset of those candidates vectors which will sum to that target vector. Viewed from this perspective, the problem can be mapped to the classic Sum-Of-Subsets (SOS) Problem which is, itself, a variant of the Knapsack Problem(8). The backtracking form of the SOS algorithm, given a positive total integer weight W and a set of n positive integers, determines all combinations of n which sum to W.

The initial problem in a straightforward implementation using Sum-Of-Subsets lies in the fact that the vector components don't necessarily start at zero. For example, if the desired target of dimension one were defined as the range 0.0 to 0.8, it would seem trivial to treat W for that dimension as 8, and find the vectors which sum to that value (after having their contributions similarly scaled by 10). Unfortunately, what if the desired target ranges from 0.4 to 0.8? The magnitude of the contribution can be scaled to 4, but then 2 vectors having contributions from 0.0 to 0.2 and 0.8 to 1.0 could be considered a solution, even though nether vector falls within the desired range at all.

The solution was to devise a schema whereby a vector's contribution to a given dimension could be represented by a single integer value that would still capture the information regarding upper and lower bounds of the covered range. Since the earliest computers decimal numbers have been represented in binary form; this binary representation is a straightforward mapping between a single scalar value and a serious of placeholders (the binary digits) which are either "off" or "on." A similar mapping was used for this implementation. To begin, the normalized range was divided into 10 subdivisions, each representing a single value of 0.1 along the total range. The choice for the number of divisions was only partially arbitrary; visualizing a span in tenths is very intuitive in the decimal system in common use, and also maps well to the idea of percentiles as encountered in test scores. This representation could easily be modified to use more (or fewer) bins as desired. Figure 3.4 depicts a single dimension of a hypothetical target vector, as well as three candidate vectors for summing to the target. Note how each vector is depicted in terms of which of the 10 subdivisions, or "bins" covered by its range of contribution.

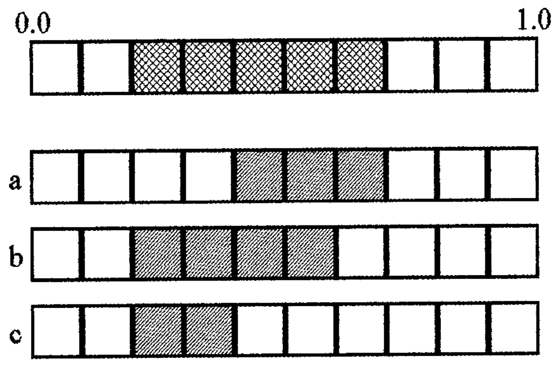


Figure 3.4 **Target Vector and Three Candidates, Using "Bin" Representation**

By inspection, the best fit of vectors to cover the target would be vectors *a* and *c*.

The key to this method of reducing a range to a single integer is using binary coding. The bins are numbered, left to right, with the powers of two in the traditional sequence. The resulting values for each bin are depicted in Figure 3.5. The bins which are included in the target range are considered "on," or binary one, while the remaining bins are "off," or binary zero. By adding the binary values of the "on" bins, one arrives at a single integer.

By following this convention, a single integer uniquely identifies precisely which bins are to be covered. As the figure shows, the sum of vectors *a* and *c* equals the value of the target vector; this representation permits direct use of the SOS algorithm to select sets of vectors which will sum to the target in the given dimension, while retaining the necessary information about upper and lower ranges.

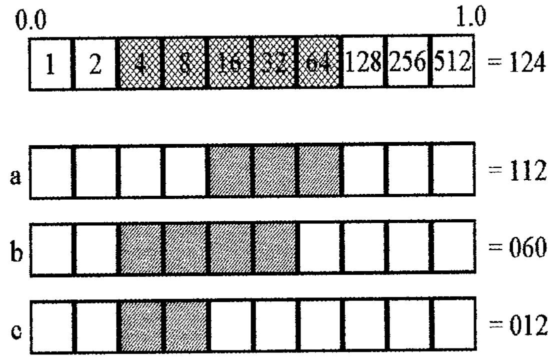


Figure 3.5 **Target Vector and Three Candidates, With Binary Values**

With this convention in mind, the components of the three sample vectors originally depicted in Figure 3.3 would be converted from pairs of real numbers to single integers containing the same information, as shown in Figure 3.6.

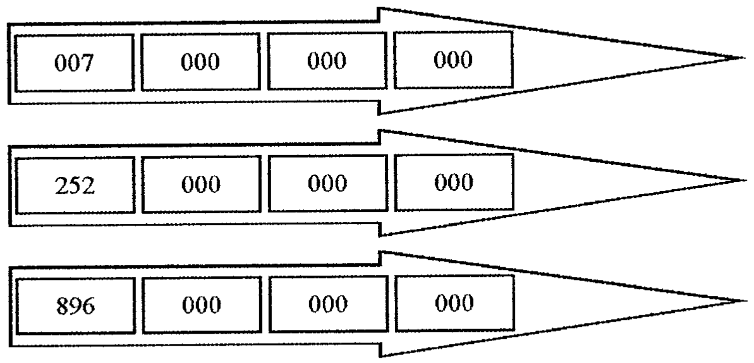


Figure 3.6 **Three Defined Concept Vectors using Binary Coded Values**

Note how the overlap between the vectors (discussed previously) translates here as the vectors summing to 1,155. Since a full span (assuming ten bins) should sum to 1,023 it's apparent that in order to allow for overlap the SOS algorithm would have to test sums against a target range rather than an exact value. In fact, the algorithm as presented provides for setting a fixed delta beyond the target value, while introducing a more automated preocedure that would iteratively expand the range some number of bins at a time until a solution were found would be a straightforward modification. In such a case, having only 10 bins might cause successive "jumps" to be too large in proportion to the overall range; even expanding the range in a single direction would mean expansion of ten per cent per iteration, which could rapidly reach the bound of the range. If greater resolution were desired the number of bins could be increased, but care must be taken not to exceed the target platform's limitations. Many systems have a limit of 32 or 64 bits which can be used to represent integer values, which imposes a limit on the number of bins one can use for this representation scheme.

3.2.2.2 Discussion of the Sum Of Subsets Approach. NOTE: The interested reader is directed to Appendix D for detailed pseudo-code of this algorithm.

The vectors are sorted in nondecreasing order for each successive dimension, so that the function *promising* can halt progress down a search path as soon as it's obvious that the next weight would exceed the limit. This continual resorting requires that each concept vector carry an additional field of information relating to its absolute position in the original array; this position will be needed to uniquely identify the vectors selected by the array *include*.

Since a greedy approach is being used, the procedure *sum_of_subsets* is allowed to terminate when it has identified the first set of vectors to satisfy the dimension in question. Once the algorithm has satisfied all n iterations, the array *include* will be set *true* for each vector which was selected. Duplicate selections are handled trivially, since setting a vector's flag to *true* multiple times in *include* has no effect on the final result. Once the algorithm terminates, the indices flagged in *include* are mapped back to the original vectors in A and included in the final list, S, which is the lesson plan.

3.2.2.3 Problems with Sum Of Subsets. The first difficulty encountered during implementation was the definition of optimality in this case. SOS, being a variant of the Knapsack Problem(8), converges to a solution which contains as many items as possible—the opposite of the desired result. This problem can be overcome by sorting the candidate components in nonincreasing order, rather than nondecreasing, and modifying the treatment of nonpromising nodes, but the resulting algorithm must include additional bounds checking. A further problem, and one which is not so readily overcome, is treatment of overlapping components. In such cases the integer sum of the components is greater than the integer value of the resultant. Although replacing addition with the bitwise OR function can mitigate this problem, it also prevents screening out vectors whose components provide no new contribution ($63 \vee 55$ is still 63, and so a vector having a component of 55 would be erroneously included in the solution set) without additional filtering.

In conclusion, SOS was rejected as the algorithm of choice for this implementation.

3.2.2.4 Set Covering Algorithm. NOTE: The interested reader is directed to Appendix E for detailed pseudo-code of this algorithm.

While revisiting the original problem, especially with a focus on dealing with overlapping vector components, the underlying misconception which led to attempting the SOS approach became clear. Rather than visualizing the target vector as a container into which lesson modules are stored, the true nature of the problem is closer to attempting to cover a given set with as few subsets as possible.

To define "cover" in this context, the original bin representation is retained. For example, a target vector component ranging from 0.2 to 0.8 would be a set containing six of the possible ten bins as members. The optimal solution would be the fewest number of lesson module vectors which, when combined, contain at least one of each of the six target bins. The total lesson plan would be, then, the set of lesson module vectors which cover all n dimensions of the target vector.

The key issues in implementing this algorithm are filtering out candidates which either lie outside the desired range or cover only elements already covered, and insuring greedy selection of candidates from F. The first issue is resolved using the same binary conversion of the bins as described in the SOS attempt. Since a single integer then uniquely describes every bin in the range, candidate selection is performed by bitwise comparison. Given target range X and a candidate F, F lies entirely within the range of X if $X \wedge F = F$. Also, given a partially-covered range U and a candidate F, F will cover at least one more element of U if $U \wedge F > 0$. Insuring greedy selection of candidates is performed by sorting the members of X by the width of their range and then selecting candidates in nonincreasing width order. This insures that each candidate selected covers at least as many elements as any candidate as yet untried.

3.3 Results

NOTE: The interested reader is directed to Appendix C for an annotated transcript of a session with the lesson plan generator.

The lesson plan generator was tested using a notional database of twenty lesson modules, five per concept. The session transcript in Appendix C lists the actual modules in detail.

The first sample execution was a trivial one, designed to insure no spurious modules were accepted outside of the selected range:

```
************************************************************
The target vector selected is:


Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0


==========================================
Selected Modules (by title):
************************************************************
```

The system correctly rejected all candidate modules, as shown above.

The next execution was designed to select a range wherein the opportunity arose to select two modules or a single one covering the same range.

```
************************************************************


The target vector selected is:


Lower Bound: 0.2 Upper Bound: 0.7
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
=========================================

Selected Modules (by title):
Feeding Four
***********************************************************
```

The system correctly selected the single, larger module which matched the target range and rejected “Feeding Three” (range of 0.4 to 0.7) and “Feeding Five” (range of 0.2 to 0.4).

The next execution was designed to show that one of the modules rejected above would, in fact, be selected under the proper cicumstances. The target range was adjusted to a range wherein “Feeding Three” was the expected selection.

```
***********************************************************

The target vector selected is:

Lower Bound: 0.4 Upper Bound: 0.7
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
=========================================

Selected Modules (by title):
Feeding Three
***********************************************************
```

As shown above, the previously rejected module was selected in this case, demonstrating that the sole criterion for the earlier rejection was, in fact, its suitability to cover the target range within the programmed selection criteria.

The next example reinforces the selection of an optimal cover by introducing another case where multiple smaller vectors cover the same range as a larger one, only this time with some overlap. In addition, the expected selection should be two lesson modules instead of just one (if the dual coverage of the same range is incorrectly handled by rejecting all modules in that range) or three (if the system incorrectly selects the two smaller, overlapping vectors instead of the single one covering more of the range).

```
**********************************************************

The target vector selected is:

Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0.8
Lower Bound: 0 Upper Bound: 0
========================================

Selected Modules (by title):
Intra-Muscular Fat One
Intra-Muscular Fat Five
**********************************************************
```

As depicted in the output above, the system rejected the smaller modules and correctly covered the target range with only two lesson modules; the optimal solution.

The final sample execution is a combination of insuring overlapping vectors are still allowed to contribute to covering the goal, while smaller ones are rejected in favor of ones providing more coverage. The expected outcome is that three out of the five possible modules related to the concept will be selected, while "Meat Juiciness Three" will be rejected for lying below the desired range and "Meat Juiciness Four" will be rejected because its contribution is inferior to that of "Meat Juiciness Five."

```
**********************************************************
```

```
The target vector selected is:


Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.3 Upper Bound: 1

=========================================


Selected Modules (by title):

Meat Juiciness One

Meat Juiciness Two

Meat Juiciness Five

***********************************************************
```

As shown in the output above, the expected three modules were chosen.

IV. Enhancements and Future Research

4.1 Enhancements

During the course of this research, numerous avenues for program enhancements came to mind, or were pointed out during discussions with faculty and fellow students. Some of those enhancements are detailed here.

4.1.1 Graphics.

Ideally, the user interface could present a graphic of the concept map for the desired domain, allowing the educator to click directly on the depicted nodes as desired and enhancing the intuitive feel of interacting with the system. In addition, generating graphical, flowchart-like output bearing some relation to the original concept map could be useful.

4.1.2 Quality of Instructional Material.

Each concept vector could carry an additional field of information: an evaluation of quality. Awarded when evaluated for inclusion into the database of concept vectors, this measurement would allow for qualitative selection of superior lesson materials in cases where more than one unit would otherwise provide the same coverage.

4.1.3 Nonlinear Coverage Within a Concept.

This research assumes blocks of coverage to be selected via an upper and lower bound, with every "bin" in between included. The method of assigning numeric values through binary coding of bins used in this system would easily permit selecting non-contiguous bins for coverage, but would introduce added complexity in terms of the user interface. Not only would the educator be required to click each individual bin desired, the interface itself would be tied to the number of bins—which might not remain constant between knowledge domains.

4.2 Recommendations for Future Research

One of the most interesting avenues of outgrowth for this system is the possibility of direct integration into an ITS. As discussed in previous chapters, units of instructional material can easily be lesson modules coded into an existing ITS. Using this sytem, it is entirely possible to have the lesson plan feed directly into the module selection for an ITS session. Further, with the close coupling this system enforces between lessons and concepts, test results could be fed back to the lesson plan generator directly, resulting in dynamic updating of the plan to accomodate needed remedial lessons in the next session.

Appendix A. Complete Concept Map

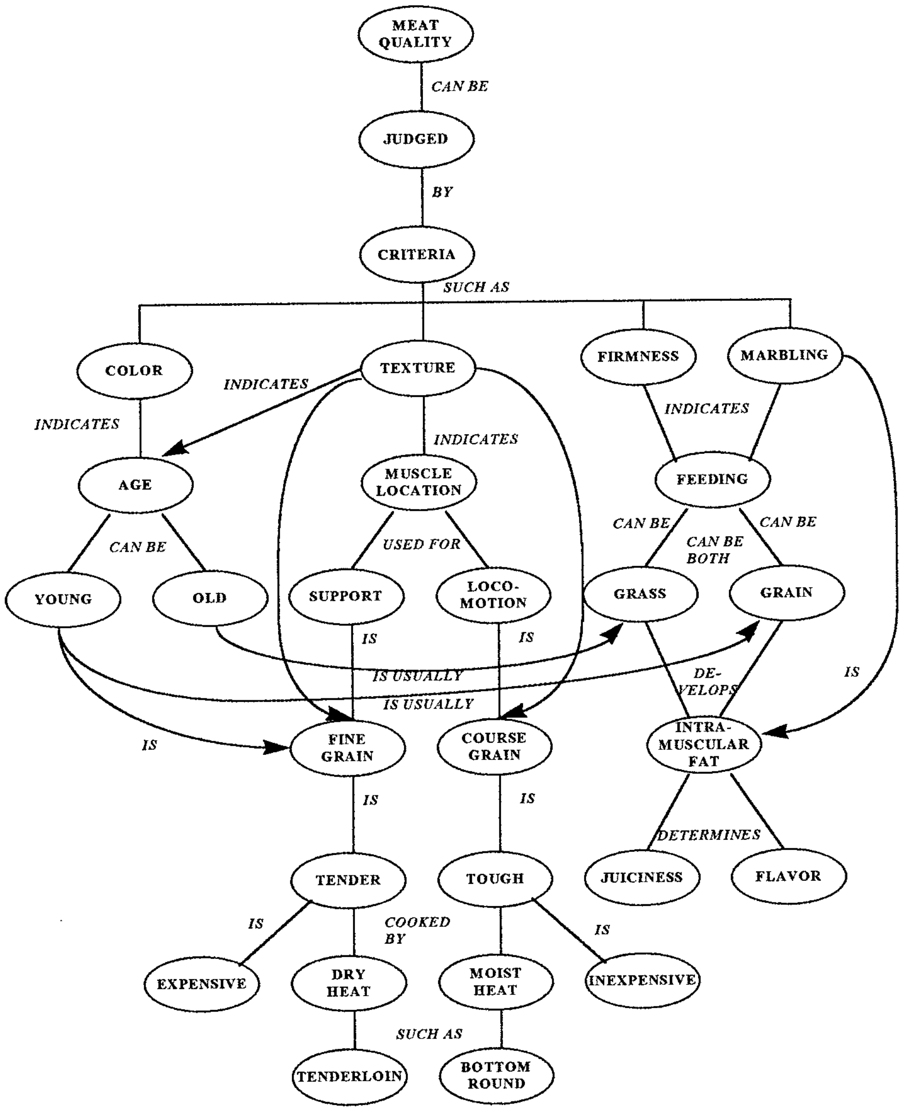
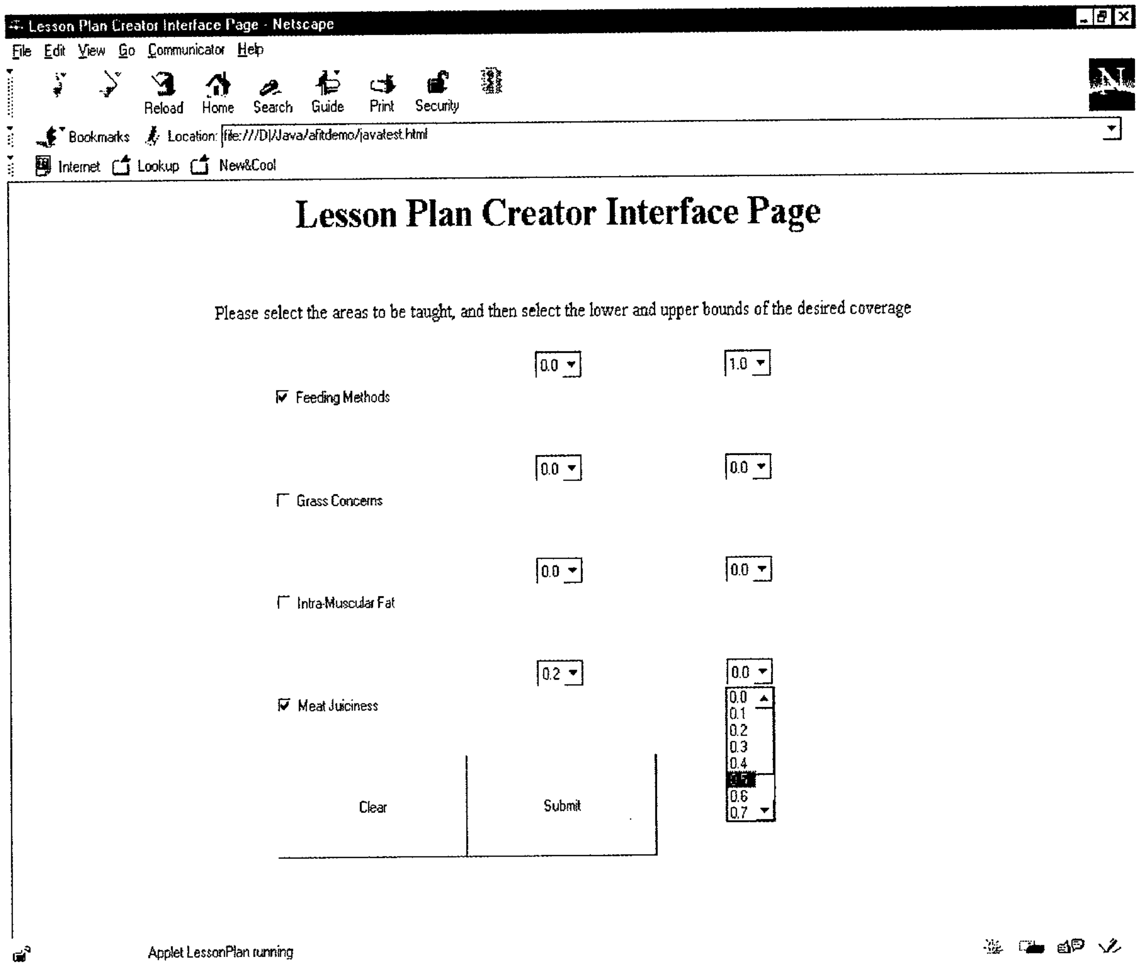


Figure A.1 **Complete Concept Map of Meat Science Course**(9)


Lesson Plan Creator Interface Page - Netscape
File Edit View Go Communicator Help
Reload Home Search Guide Print Security
Bookmarks Location: file:///D|/Java/afitdemo/javatest.html
Internet Lookup New&Cool

Lesson Plan Creator Interface Page

Please select the areas to be taught, and then select the lower and upper bounds of the desired coverage

☑ Feeding Methods	0.0	1.0
☐ Grass Concerns	0.0	0.0
☐ Intra-Muscular Fat	0.0	0.0
☑ Meat Juiciness	0.2	0.0

0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Clear
Submit
Applet LessonPlan running

Figure B.1 **Prototypical WWW Interface**

Appendix C. Sample Session

The following sample session is based on a notional database of twenty lesson modules based on the four-concept subset of the Meat Science example depicted in Figure 3.1. The initial output is for illustrative purposes only, and is designed to provide documentation of the twenty vectors in the lessons database.

The session begins with the instructor selecting all four concepts, with complete coverage of each (selected range from 0.0 to 1.0 in each case).

Initial print-out of all twenty lesson modules:

```
***********************************************************
*                                                         *
*               Dynamic Lesson Plan Generator             *
*                                                         *
***********************************************************

The complete lesson module database is:

Dimensionality: 4
Number of Modules: 20
Index: 1 // Title: Feeding One
Lower Bound: 0.5 Upper Bound: 1
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
=========================================
```

****

NOTE: Throughout this sample session the above line of asterisks will appear immediately before and after inserted author comments to distinguish them from actual program output.

Key to module descriptions:

Dimensionality The number of nodes (dimensions) in the vector space.

Number of Modules The total number of modules defined in the database.

Index An index for locating the given module within the database. Can be an array index, a hashing key, or any other appropriate value. Note this index is intended for a machine use.

Title The actual title of the given lesson module. Note this is intended as a human-readable index for locating the module, but could be useful for string oriented machine searches.

Lower Bound The lowest level covered by the given module along the normalized scale.

Upper Bound The highest level covered by the given module along the normalized scale.

****

```
Index: 2 // Title: Feeding Two
Lower Bound: 0 Upper Bound: 0.3
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
========================================
Index: 3 // Title: Feeding Three
Lower Bound: 0.4 Upper Bound: 0.7
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
========================================
Index: 4 // Title: Feeding Four
Lower Bound: 0.2 Upper Bound: 0.7
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
========================================
Index: 5 // Title: Feeding Five
Lower Bound: 0.2 Upper Bound: 0.4
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
========================================
Index: 6 // Title: Grass Concerns One
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0.5 Upper Bound: 0.8
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
========================================
Index: 7 // Title: Grass Concerns Two
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0.1 Upper Bound: 0.4
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
========================================
Index: 8 // Title: Grass Concerns Three
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0.2 Upper Bound: 0.5
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
========================================
Index: 9 // Title: Grass Concerns Four
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0.2
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0

=========================================

Index: 10 // Title: Grass Concerns Five

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.7 Upper Bound: 1

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

=========================================

Index: 11 // Title: Intra-Muscular Fat One

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0.4

Lower Bound: 0 Upper Bound: 0

=========================================

Index: 12 // Title: Intra-Muscular Fat Two

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.8 Upper Bound: 1

Lower Bound: 0 Upper Bound: 0

=========================================

Index: 13 // Title: Intra-Muscular Fat Three

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.5 Upper Bound: 0.8

Lower Bound: 0 Upper Bound: 0

=========================================

Index: 14 // Title: Intra-Muscular Fat Four

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.4 Upper Bound: 0.6
Lower Bound: 0 Upper Bound: 0

=========================================

Index: 15 // Title: Intra-Muscular Fat Five

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.4 Upper Bound: 0.8

Lower Bound: 0 Upper Bound: 0

=========================================

Index: 16 // Title: Meat Juiciness One

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.5 Upper Bound: 0.8

=========================================

Index: 17 // Title: Meat Juiciness Two

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0.3 Upper Bound: 0.7

=========================================

Index: 18 // Title: Meat Juiciness Three

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0.5

=========================================

Index: 19 // Title: Meat Juiciness Four

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0

Lower Bound: 0 Upper Bound: 0
Lower Bound: 0.7 Upper Bound: 0.9
=========================================
Index: 20 // Title: Meat Juiciness Five
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0.7 Upper Bound: 1
=========================================
```

****

The following portion of output is an echo to the instructor to verify the target vector about to be used as the goal of a lesson module search.

****

```
The target vector selected is:

Lower Bound: 0 Upper Bound: 1
Lower Bound: 0 Upper Bound: 1
Lower Bound: 0 Upper Bound: 1
Lower Bound: 0 Upper Bound: 1
=========================================
```

****

The following portion of output is the listing (by title) to the instructor of the lesson modules selected by the program. Note that each title bears an associated index value which can be passed to an existing ITS in the desired format to enable the ITS to automatically select the appropriate modules for a session.

****

```
Selected Modules (by title):
Feeding One
Feeding Two
Feeding Four
Grass Concerns One
Grass Concerns Two
Grass Concerns Three
Grass Concerns Four
Grass Concerns Five
Intra-Muscular Fat One
Intra-Muscular Fat Two
Intra-Muscular Fat Five
Meat Juiciness One
Meat Juiciness Two
Meat Juiciness Three
Meat Juiciness Five

     *****************************************
```

It's interesting to note some aspects of the results from this initial session. For example, the modules "Feeding Three" and "Feeding Five" were not selected, because their coverage is entirely handled by module "Feeding Four," which was selected. This reflects the assumption that covering the goal using the fewest possible number of modules is optimal. Other modules in the concepts of "Intra-Muscular Fat" and "Meat Juiciness" were similarly rejected in favor of single modules providing the same coverage.

As the student progresses through the entire session, each concept would be tested by the ITS and the scores retained and mapped against the concept scales. Ideally, as in this notional session, the tests for each module are matched to the normalized scale used to measure progress within the concept. Once the session is completed, the ITS identifies the range(s) within each concept wherein the student failed to perform, and sends these values back to the lesson plan generator to select appropriate remedial modules.

```
        ****************************************


The target vector selected is:

Lower Bound: 0.2 Upper Bound: 1
Lower Bound: 0 Upper Bound: 0
Lower Bound: 0.5 Upper Bound: 1
Lower Bound: 0.7 Upper Bound: 1
=========================================

        ****************************************
```

In this example, the student performed very poorly during the "Feeding" concept scoring only 20 per cent, but mastered the concept of "Grass Concerns." The other two concepts resulted in a mixed level of success, 50 and 70 per cent, respectively. In this example a required threshold of 80 per cent is assumed in order to avoid remediation, and so three concepts are to be covered in the next session.

```
        ****************************************


Selected Modules (by title):
Feeding One
Feeding Four
Intra-Muscular Fat Two
Intra-Muscular Fat Three
Meat Juiciness Five

        ****************************************
```

Again, some aspects of these results are of interest. For example, "Feeding One" and "Feeding Four" are retained as providing complete coverage using the fewest vectors. Since the only criteria available to the lesson plan generator is level of coverage, this is an expected result. Were additional criteria added, such as a scale for weighting the quality of a module against some metric, remediation selection could be expected to vary as the quality metric were adjusted to reflect demonstrated effectiveness.

The module "Intra-Muscular Fat Three" is selected for the remedial session while "Intra-Muscular Fat Five" is dropped. This selection reflects the fact that the smaller level of coverage provided by the new module maps more exactly to the reduced goal, and demonstrates the assumption that selecting modules which provide coverage outside the goal is not desireable.

Note, too, that module "Meat Juiciness Five" is selected, and so the module "Meat Juiciness Four" is not. This selection again reflects the greedy nature of the algorithm, where the largest coverage was selected first and subsequent smaller modules which do not contribute by covering previously uncovered bins are rejected.

****

end of run

Appendix D. Sum Of Subsets Pseudo-Code(8)

Problem: Given a target concept vector V of size n, a delta d, and an array A of m concept vectors in the same knowledge domain as V, derive a subset S from A consisting of k vectors which sum to V plus d.

Inputs: V, A, m, n.

Outputs: S.

```
/*******************************************************/
procedure generate_plan (in concept_vector:  V;
                         in vector_list:     A;
                         in integer:         m, n, d;
                         in out vector_list: S);


integer i, j, W, total;
boolean include[m];


begin
    for(i := 1 to m) do
        include[i] := false;
    endfor
    initialize_to_null(S);
    for(i := 1 to n) do
        total := 0;
        for(j := 1 to m) do
            total := total + A[i][j];
        endfor
        W := V[i] + 2^(i + d); //target plus delta in bins
        sort_array_nondecreasing_on_i(A, i); //must recall prior position
        sum_of_subsets(0, 0, total, A, i, W, include);
    endfor
    for(i := 1 to m) do
        if include[i] = true then
            add_to_S(S, A[i]);
        endif
    endfor
end;

/*********************************************************/
procedure sum_of_subsets (in integer:        index;
                          in integer:        weight;
                          in integer:        total;
                          in vector_list:    A;
                          in integer:        i;
                          in integer:        W;
                          in out boolean[]: include);

begin
    if promising(index, weight, total, A, i, W) then
        if weight = W then
            return;
        else
            include[index + 1] := true;
            sum_of_subsets(index+1, weight+A[i][index +1],
                           total-A[i][index +1], A, i, W, include);
            include[index + 1] := false;
            sum_of_subsets(index+1, weight,
                           total-A[i][index +1], A, i, W, include);
        endif
    endif
end;
```

```
/**********************************************************/

function promising(in integer:     index
                   in integer:     weight;
                   in integer:     total;
                   in vector_list: A;
                   in integer:     i;
                   in integer:     W): boolean;


begin
    promising := (weight + total >= W) and
                 ((weight = W) or (weight + A[i][index + 1] <= W));
end;

/**********************************************************/
```

Appendix E. Set Covering Pseudo-Code(11)

The revised algorithm is as follows (procedure generate_plan omitted):

Problem: Given a target concept vector X of size n, and an array F of m concept vectors in the same knowledge domain as X, derive a minimal subset C from F consisting of k vectors which cover X.

Inputs: X, F, m, n.

Outputs: C.

```
/*********************************************************/
procedure greedy_set_cover(out set C)


begin
    U := X; //tempory working copy
    C := NULL;
    index := 1;
      while((U > 0) && (index <= masterDB.m)) do
          if( F[index] covers only elements of X and
                  F[index] covers at least one elemet of U then
              U := U - F[index];
              C := C + F[index];
          endif;
          increment index;
      endwhile;
      if U not empty then
          print error message;
          return;
      endif;
end;

/*********************************************************/
```

Appendix F. Java Source Code

```
//package thesis;

// convec.java

import java.lang.*;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.File;
import java.io.IOException;

//****************************************************************************
class Component {
 private float begin;
 private float end;
 private float size;
 private int binaryValue;

//----------------------------------------------------------------------------
 public Component(float beginValue, float endValue) {
  if(((beginValue < 0)||(beginValue > 1)) || ((endValue < 0)||(endValue > 1))
   || (endValue < beginValue)) {
  System.out.println("Bad Component constructor values #1!");
  }
  else {
    begin = beginValue;
    end = endValue;
    size = endValue - beginValue;
    binaryValue = binval(begin, end);
  }
 }

//---------------------------------------------------------------------------
 public float begin() {
  return begin;
 }

//---------------------------------------------------------------------------
 public float end() {
  return end;
 }

//---------------------------------------------------------------------------
 public float size() {
  return size;
 }

//---------------------------------------------------------------------------
 public int binaryValue() {
  return binaryValue;
 }

//---------------------------------------------------------------------------
 public int binval(float start, float finish) {

  int temp = 0;
  float a = start * 10;
  float b = finish * 10;
 for(int i = (int) a; i < (int) b; ++i)
   {
       temp += Math.pow(2, i);
   }

 return temp;
}

//------------------------------------------------------------------------
public void beginSet(float beginValue) {
 if((beginValue >= 0)&&(beginValue <= 1)&&(beginValue <= end)) {
   begin = beginValue;
   size = end - begin;
   binaryValue = binval(begin, end);
 }
 else {
 System.out.println("Tried to set bad begin in beginSet!");
 }
}

//------------------------------------------------------------------------
public void endSet(float endValue) {
 if((endValue >= 0)&&(endValue <= 1)&&(begin <= endValue)) {
   end = endValue;
   size = end - begin;
   binaryValue = binval(begin, end);
 }
 else {
 System.out.println("Tried to set bad end in endSet!");
```

```
  }
 }

//-----------------------------------------------------------------------------
 public String toString() {
  return new String("("+String.valueOf(begin)+","+String.valueOf(end)+")");
 }
}

//****************************************************************************
class LessonVector {

  Component components[];
  float length;
  int index;
  String title;

//-----------------------------------------------------------------------------
  public LessonVector(int n) {
    components = new Component[n];
    length = 0;
  }

//-----------------------------------------------------------------------------
  public void SetComponent(int j, float a, float b) {
    components[j] = new Component(a, b);
    length += components[j].size();
  }
}
```

```
//**************************************************************************
class LessonsDataBase {

  int m, n;
  LessonVector lessonList[];

//----------------------------------------------------------------------------
  public LessonsDataBase(String s) throws IOException {

    String inputBuffer;
    Integer intBuffer;
    Float   floatBuffer;
    float a, b;

    File file = new File(s);
    FileInputStream inFile = new FileInputStream(file);
    DataInputStream inStream = new DataInputStream(inFile);
    inputBuffer = inStream.readLine();
    intBuffer = Integer.valueOf(inputBuffer);
    n = intBuffer.intValue();
    inputBuffer = inStream.readLine();
    intBuffer = Integer.valueOf(inputBuffer);
    m = intBuffer.intValue();
    lessonList = new LessonVector[m + 1];
    for(int i = 1; i <= m; ++i) {
      lessonList[i] = new LessonVector(n + 1);
      lessonList[i].index = i;
      lessonList[i].title = inStream.readLine();
      for(int j = 1; j <= n; ++j) {
        inputBuffer = inStream.readLine();
        floatBuffer = Float.valueOf(inputBuffer);
        a = floatBuffer.floatValue();
        inputBuffer = inStream.readLine();
        floatBuffer = Float.valueOf(inputBuffer);
        b = floatBuffer.floatValue();
        lessonList[i].SetComponent(j, a, b);
      }
    }
  }
}

//****************************************************************************
class TargetVector {

  LessonVector target;

//----------------------------------------------------------------------------
  public TargetVector(String s, int n) throws IOException {

    String inputBuffer;
    Integer intBuffer;
    Float   floatBuffer;
    float a, b;

    File file = new File(s);
    FileInputStream inFile = new FileInputStream(file);
    DataInputStream inStream = new DataInputStream(inFile);
    target = new LessonVector(n + 1);
    for(int i = 1; i <= n; ++i) {
        inputBuffer = inStream.readLine();
        floatBuffer = Float.valueOf(inputBuffer);
        a = floatBuffer.floatValue();
        inputBuffer = inStream.readLine();
        floatBuffer = Float.valueOf(inputBuffer);
        b = floatBuffer.floatValue();
        target.SetComponent(i, a, b);
    }
  }
}

//****************************************************************************
class run {

    static boolean include[];
    static boolean finalList[];
    static boolean tempList[];
    static LessonsDataBase masterDB;
    static TargetVector goalVector;
    static int W;
    static int currentIndex;
    static int currentDimension;
    static int currentDim[];

//-----------------------------------------------------------------------------
  public static void main(String args[]) throws IOException {

    int i, j;
    int total;

    masterDB = new LessonsDataBase("testdb.txt");
```

```
// goalVector in production implementation would be recieved
// from WWW interface and/or test results supplied by ITS
goalVector = new TargetVector("testvec1.txt", masterDB.n);
finalList = new boolean[masterDB.m + 1];
tempList = new boolean[masterDB.m + 1];
currentDim = new int[masterDB.m + 1];


System.out.println(" ");
System.out.println("*************************************************************");
System.out.println("*                                                           *");
System.out.println("*               Dynamic Lesson Plan Generator               *");
System.out.println("*                                                           *");
System.out.println("*************************************************************");
System.out.println(" ");
System.out.println("The complete lesson module database is:");
System.out.println(" ");
printDB(masterDB);
System.out.println(" ");
System.out.println("The target vector selected is:");
System.out.println(" ");
printVec(goalVector);
System.out.println(" ");


for(i = 1; i <= masterDB.m; ++i) {
    finalList[i] = false;
}


for(i = 1; i <= masterDB.n; ++i) {


    for(j = 1; j <= masterDB.m; ++j) {
            tempList[j] = false;
        }

        W = goalVector.target.components[i].binaryValue();
        currentIndex = masterDB.lessonList[i].index;
        currentDimension = i;

        quicksort(masterDB, 1, masterDB.m, i);
        for(j = 1; j <= masterDB.m; ++j) {
            currentDim[j] = masterDB.lessonList[j].components[i].binaryValue();
        }

        greedySetCover(1);
    }

    System.out.println("Selected Modules (by title):");
    for(i = 1; i <= masterDB.m; ++i) {
        if(finalList[i] == true) {
            for(j = 1; j <= masterDB.m; ++j) {
                if(masterDB.lessonList[j].index == i) {
                    System.out.println(masterDB.lessonList[j].title);
                    break;
                }
            }
        }
    }

  }

//---------------------------------------------------------------------------
```

```
public static void greedySetCover(int index)
                                        throws IOException {

    int U = W;
    while((U > 0) && (index <= masterDB.m)) {
        if(((currentDim[index] | W) == W) && ((currentDim[index] & U) > 0)) {
            U = (U & ~currentDim[index]);
            tempList[index] = true;
        }
        index++;
    }
    if(U > 0) {
        System.out.println("No cover found for dimension "+currentDimension);
        return;
    }
    for(int zz=1; zz <= masterDB.m; ++zz) {
        finalList[masterDB.lessonList[zz].index] |= tempList[zz];
    }
}

//------------------------------------------------------------------------
public static void printDB(LessonsDataBase A) throws IOException {

  System.out.println("Dimensionality: "+A.n);
  System.out.println("Number of Modules: "+A.m);
  for(int i = 1; i <= A.m; ++i) {
     System.out.print("Index: "+A.lessonList[i].index+" // ");
     System.out.println("Title: "+A.lessonList[i].title);
     for(int j = 1; j <= A.n; ++j) {
       System.out.println("Lower Bound: "+A.lessonList[i].components[j].begin()+
                        " Upper Bound: "+A.lessonList[i].components[j].end());
      }
      System.out.println("==========================================");
    }
  }

//------------------------------------------------------------------------------
  public static void printVec(TargetVector A) throws IOException {

    for(int j = 1; j <= masterDB.n; ++j) {
        System.out.println("Lower Bound: "+A.target.components[j].begin()+
                           " Upper Bound: "+A.target.components[j].end());
    }
    System.out.println("==========================================");
  }

//------------------------------------------------------------------------------
  public static void quicksort(LessonsDataBase A, int p, int r, int k) throws IOExceptic

    if (p < r) {
      int q = partition(A, p, r, k);
      quicksort(A, p, q, k);
      quicksort(A, q+1, r, k);
    }
  }

//------------------------------------------------------------------------------
  public static int partition(LessonsDataBase A, int p, int r, int k) throws IOExceptior

    float x = A.lessonList[p].components[k].size();
```

```
        int i = p - 1;
        int j = r + 1;
        LessonVector temp;

        while(true) {
            do {
                j = j - 1;
            } while(A.lessonList[j].components[k].size() < x);
            do {
                i = i + 1;
            } while(A.lessonList[i].components[k].size() > x);
            if(i < j) {
                temp = A.lessonList[j];
                A.lessonList[j] = A.lessonList[i];
                A.lessonList[i] = temp;
            }
            else {
                return j;
            }
        }
    }

//------------------------------------------------------------------------
}
```

Bibliography

1. Ahlberg, Christopher and Ben Schneiderman. "The Alphaslider: A Compact and Rapid Selector." From the Human-Computer Interaction Laboratory and Institude for Systems Research, 1997.

2. Alberto Regis, Pier Giorgio-Albertazzi and Ezio Roletto. "Concept Maps in Chemistry Education," *Journal of Chemical Education*, *73*(11) (1996).

3. Auer, John W. *Linear Algebra, With Applications*. Prentice-Hall, 1991.

4. Donald Ely, Alan Januszewski and Glenn Le Blanc. *Trends and Issues in Educational Technology 1988*. Syracuse University Press, 1988.

5. Guthrie, E. R. "Conditioning as a Principle of Learning," *Psychological Review*, *37* (1930).

6. HQ AETC/XORR. *AF HANDBOOK 36-2235*, 1984.

7. J. S. Brown, A. Collins and S. Duguid. "Situated Cognition and the Culture of Learning," *Educational Researcher*, *18*(1) (1989).

8. Jr., Freeman A. Kilpatrick. *A Generic Intelligent Architecture for Computer-Aided Training of Procedural Knowledge*. PhD dissertation, Air Force Institute of Technology, 1996.

9. Kabrisky, Matthew. "The Promise of an Artificial Intelligence Future." From IEEE meeting in Dayton, OH, 1983.

10. Kozen, Dexter C. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.

11. Mano, M. Morris. *Computer System Architecture* (Second Edition). New Jersey: Prentice-Hall, Inc., 1982.

12. Neapolitan, Richard and Kumarss Naimipour. *Foundations of Algorithms*. Heath, 1996.

13. Novak, Joseph and D. Bob Gowin. *Learning How to Learn*. Cambridge University Press, 1984.

14. O'Shea, Tim and John Self. *Learning and Teaching with Computers–Artificial Intelligence in Education*. Prentice-Hall, 1983.

15. Shute, Valerie J. *Individual Differences In Learning From an Intelligent Discovery World: Smithtown*. Technical Report AFHRL-TP-89-57, Manpower and Personnel Division, Brooks Air Force Base, 1990.

16. Shute, Valerie J. and Josepf Psotka. *Intelligent Tutoring Systems: Past, Present, Future*. Technical Report AL/HR-TP-1994-0005, USAF, Armstrong Laboratory, 1994.

17. Shute, V.J., et al. "Modeling Practice, Performance, and Learning." *Simulation-Based Experiential Learning* edited by D.M. Towne, et al., 133–145, Berlin:Springer-Verlag, 1992.

18. Thomas Cormen, Charles Leiserson and Ronald Rivest. *Introduction to Algorithms.* McGraw Hill, 1995.

19. Thompson, Jeremy E. *Student Modeling in an Intelligent Tutoring System.* MS thesis, Air Force Institute of Technology, 1996.

20. Thorndike, E. *The Fundamentals of Learning.* Teachers College Press, 1932.

21. Toshiyuki Asahi, David Turo and Ben Schneiderman. "Using Treemaps to Visualize the Analytic Heirarchy Process." *Proceedings of the Conference on Human Factors in Computer Systems.* 1995.

22. Winston, Patrick Henry. *Artificial Intelligence* (Third Edition). Addison-Wesley, 1993.

Vita

Captain Mark L. Dyson was born February 15, 1961, in Jacksonville, Florida. He graduated from Pensacola High School, Pensacola, Florida, in June 1978. He enlisted in the US Air Force, entered Basic Military Training in January 1979, and then attended the Defense Language Institute in Monterey, California. After studying the Korean language for nearly 18 months, he graduated with honors, was married to the former Debrah Anderson in July 1980, and then was assigned to the Electronic Security Command, stationed on Osan AB, Republic of Korea. Following two tours overseas, Mark returned to begin a series of special duty assignments with the Compass Call airborne electronic warfare platform. He served two tours at Davis-Monthan AFB, Arizona, separated by a return to the Defense Language Institute where he studied German and Russian. From Arizona, Mark was sent to help form the first Compass Call squadron to be permanently deployed overseas, at Sembach AB, Germany. While serving at Sembach he was accepted into the Airman's Education and Commissioning Program, and returned to the US to complete his Bachelor of Science in Computer and Informational Sciences at the Ohio State University.

Following award of his B.S., Mark earned his commision through the Officer Training Group in Lackland, Texas, and then attended Basic Communication-Computer Officer Training at Keesler AFB, MS. Following completion of this training, Lt Dyson was assigned to the Standard Systems Group at Gunter Annex, Maxwell AFB, AL, where he spent nearly four years conducting modeling and simulation of large scale computer systems and evaluating new technologies for possible inclusion into existing AF projects. Captain Dyson was selected to attend the Air Force Institute of Technology (AFIT), and began his studies in June 1996. Upon graduation from AFIT in December 1997, Captain Dyson will be assigned to Rome Laboratory, in Rome, NY, where he'll be joining the team helping develop technology for human-computer interaction in the Air Force.

Permanent address: 212 Marine Drive
Warrington, Florida 32507

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE CONCEPT VECTORS: A SYNTHESIS OF CONCEPT MAPPING AND MATRICES FOR KNOWLEDGE REPRESENTATION IN INTELLIGENT TUTORING SYSTEMS	5. FUNDING NUMBERS
6. AUTHOR(S) Mark L. Dyson, Captain, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/97D-07
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) USAF Phillips Laboratory/VTS Captain Freeman A. Kilpatrick, Ph.D. 3550 Aberdeen Avenue SE Kirtland AFB, NM 87117-5776	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A review of the literature relating to intelligent tutoring systems (ITS) reveals that the bulk of research to date is focused on the student, and on methods for representing the knowledge itself. From student models to learning schemas to presentation methods, comparatively little attention has been paid to the problem of educators attempting to build viable lesson plans for use in an ITS environment--yet when this problem is addressed in the literature, it is recognized as a potentially daunting one. This thesis addresses the problem of ITS lesson plan development by proposing a practical, computable approach for knowledge engineering that is based on proven classroom methods. The document then details a system for dynamically creating lesson plans from a knowledge base created under the described methodology, using already-established algorithms of proven tractability, and then discusses how this system can be integrated into existing and future ITS design.

14. SUBJECT TERMS Intelligent Tutoring Systems, Artificial Intelligence, JAVA, Education, Concept Mapping, Concept Vectors	15. NUMBER OF PAGES 66
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	Contract	**PR**	Project
G	Grant	**TA**	Task
PE	Program Element	**WU**	Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	See authorities.
NASA	See Handbook NHB 2200.2.
NTIS	Leave blank.

Block 12b. Distribution Code.

DOD	Leave blank.
DOE	Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	Leave blank.
NTIS	Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.